Malware Detection Assignment Report

By Breaking Code
Ayush Sawlani (IMT2018014)
Ayush Mishra (IMT2018013)
Aarushi Goenka (IMT2018001)

I. Introduction

In this day and age everything is getting digitalised. From children of age as young of 5 to people of the age 90, we see everyone associated with some kind of technology. In times like these, it is more important than ever to make cyber security our top priority. Once a computer is infected by malware, it becomes very easy for criminals and other enterprises to steal important information and cause other harms. The rise of the internet and growth in the malware industry made it almost impossible for humans to create detection rules and manually predict malware. Malware detection is an extremely important topic, and hence various enterprises and individuals have worked on developing techniques to predict malware occurrences and create new and advanced protection technologies. Microsoft in 2019 took this problem seriously and hosted a competition to encourage people in the data science industry to come up with techniques to help protect more than a billion machines from damage before it happens.[1]

II. Dataset

The dataset used in the project is a part of the Microsoft Malware Prediction's dataset. The data containing these properties and machine infections was generated by combining heartbeat and threat reports collected by Microsoft's endpoint protection solution, Windows Defender.[1] Each data point originally had 82 columns. The train data consists of 567730 data points and the test data consists of 243313 data points. The problem is a binary classification problem, which means that a machine can be labelled to either to have a detection or to not have one, which is denoted by the last 'HasDetections' column, which is also our target column.

III. Pre-processing

A. Feature Selection

For feature selection we observed that there were categorical features which had more than 99 % of data in one category and also had some null values in them. As the data was highly skewed, we thought that there won't be any amount of useful learning happening from these. So we calculated the percentage occurrence of mode for each feature and also the percentage of null values and summed them up for each feature. Then we eliminated those features whose "null percentage + mode percentage" was greater than 98 percent. Finally we got some 64 features as our useful features. For example, 'Census IsWIMBootEnabled' had 100 percent null+mode. 'IsBeta' had 99.99 percent skewed data and so they were left out.

B. Data Cleaning

For the majority of the columns we filled the null values with the mode of the column and we also did some manipulations in some of the columns where we replaced the category with lower count and merged it into a larger category because they had similar traits.

For example in CensusMDC2FormFactor , we combined SmallTablet and LargeTablet into one bigger category called as Tablet and also combined SmallServer,MediumServer and LargeServer into one category called Server, we did this for few other columns as well such as ProductName and SmartScreen.

For one specific column CityIdentifier we decided to fill the Null Values by grouping them with CountryIdentifier because we figured out they should be having a relation between them.

C. Encoding Data

When we were trying to encode the categorical columns, we could not come up with any single type of encoding for our columns, hence decided to look at each column individually. Here are a few things we considered while trying to think of encoding for the columns:

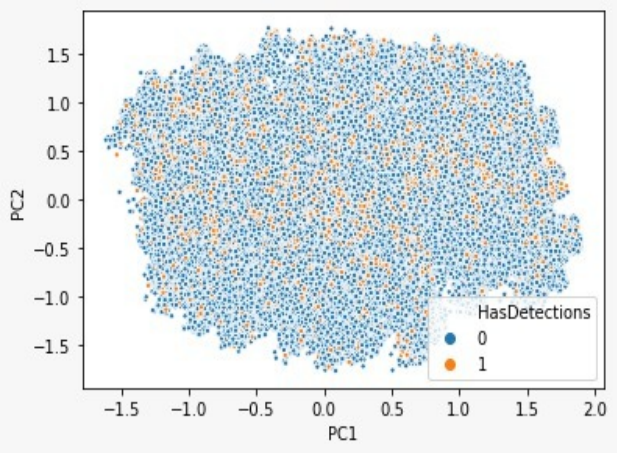
1) There were some columns, like AppVersion, EngineVersion, etc, which had a sense of time to them. Any column which told about versions of some software might follow some trend through time. Older software might be more or less prone to malware as compared to the latest versions. Hence we did label encoding on such columns to preserve the order of the versions.
2) If a particular software has many users, a hacker is more likely to attack it as his gain will be much higher if he is able to hack a popular software rather than a software which is used less. Similarly, a hacker might be more likely to hack in a city/country which has more residents. Hence we decided to go with frequency encoding for some features like CityIdentifier, Processor, etc.[2]

According to the above observations, we decided to divide most of our columns into frequency or label encoding. For most of the binary columns, we just converted them to numeric.

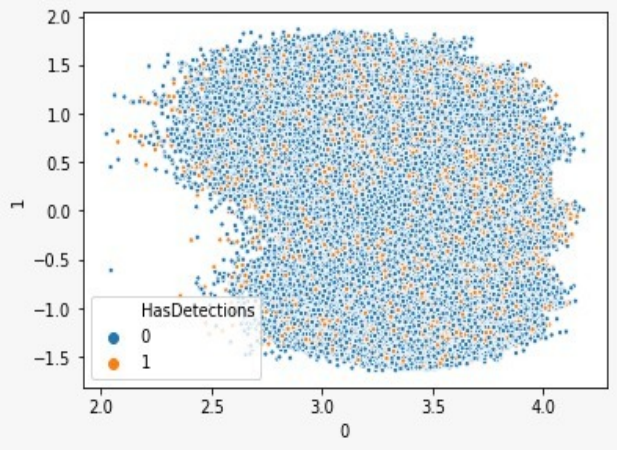
IV. PCA and other analysis

Techniques such as PCA and SVD helps us visualize how the points in N-dimensional space will look like in a 2-d Plane.[3][4]

Below is the PCA plot.



Below is the SVD plot.



As we can see none of these representations show a nice linear boundary, the algorithms which give linear separators like logistic regression or SVM won't work well. Hence we decided to use decision trees based algorithms.

V. Handling Imbalanced Data

As we saw earlier, our data has an imbalance with only 15 % of our training data belonging to the malicious class. To tackle this imbalance, We tried various methods but all in vain as none of them helped us in increasing our score.

We started with SMOTE but as we found previously from PCA, our data was not linearly separable which is why SMOTE performed poorly. We did not try any undersampling as our dataset was already quite less and undersampling would have further decreased our score. We gave random oversampling a try and it gave a slight increase in the score but still less than our best Lightgbm model. Although in the end we took an ensemble of a Lightgbm model with oversampling done and one lightgbm model without oversampling and we found it to work very well.

We next tried to work with the various ensemble methods in imblearn. We used BalancedBaggingClassifier, BalancedRandomForestClassifier and EasyEnsemble over xgboost and adaboost but found that it consumed a lot of time and the score was less than what we were getting with lightgbm[5][6][7], hence I decided to drop the idea of any sampling.

VI. Model Selection

We experimented with quite a few models , which were mostly tree-based algorithms because in the PCA analysis we could see that there was no clear linear boundary in our target column.

Here are the algorithms that we tried for this project:

1) Random Forest Classifier - This was the first model that we tried on our dataset, we wanted to start off with a basic tree model that all three of us knew ,so we went ahead with it .We achieved a highest score of 0.70509 after tuning the hyperparameters ,which although was nowhere near our best score but it gave us a good head start and confidence ,since we crossed the 0.70 barrier from the first model itself.[7]
2) XG Boost - This was the second model that we tried on our data set , this is another tree based algorithm but its run time was quite slow even with lower parameters of tree depth and number of trees but it did give us better results than the Random Forest Classifier with our highest being 0.71214,this helped us cross the 0.71 barrier but it was taking a lot of time to run so we decided to move ahead with other models.[8][9]
3) Light Gradient Boosting Machine - This was the third model that we tried on our data set. This was by far our best performing model and also the fastest running model out of all the models that we had tried or were to try in future. We tried various combinations of hyperparameters and even performed Randomized CV search on hyperparameters. The best result that we got out of it was 0.72241, but it also played a major role when we performed ensembling and got our best result of 0.72329.[10]
4) CatBoost - This was another model that we tried on our dataset. This is again another trees based algorithm and also performed quite well when we were locally testing

our data giving scores over 0.714 but since we already had much better scores using Light GBM, we never went ahead submitting any solution, much like Light GBM even this was faster than Random Forest Classifier and XG Boost but was slower than Light GBM.[11]

5) AdaBoost - This was another model we tried on our data specifically for our ensembling procedure because Adaboost is a boosting technique and is often used with other models in the ensemble method. This was by far the slowest of all the models we tried, even for the small parameters it took a huge amount of time.[12]

VII. HyperParameter Tuning

We experimented with a lot of different models/classifiers and finding the correct set of parameters for each of those was not an easy task because tuning parameters through grid/randomized CV search would take hours, so we performed search only for Light GBM.

1) For Random Forest Classifier we managed to get our best results after playing around with parameters and using simple plain logic of keeping the number of trees and the learning rate in proportion and keeping the maximum depth of our tree in correlation to the number of features we have , we managed to get our best score at : Number of Trees = 300, max depth = 25 and minimum sample split =100.
2) For XGBoost which was again was a very slow running algorithm, we decided to just play around with parameters manually and using logic and experience from the hyperparameters of Random Forest Classifier and we got our best score at : Number of trees =500,learning rate=0.1 ,max depth=10 and min child weight = 50.[14]
3) Now for Light GBM , since we knew this was going to be our best model which would yield the best result we decided to do a search on our parameters specifically on three parameters of learning rate ,the maximum depth and the minimum child weight. After doing a Randomized Search on these parameters we got their best values of learning rate of 0.006 ,the maximum depth of 10 and minimum child weight of 50 .[15] After we had these values we just used binary search on the number of trees ,number of leaves and colsamplebytree manually and found the best results to be at 4500 , 511 and 0.30 respectively.
4) Now for CatBoost we decided to go with the same parameters as Light GBM and for AdaBoost since it was a very slow algorithm and we only needed it for our Ensembling method we kept the parameters relatively low at number of trees to be 500 and learning rate to be at 0.1.

Our best score without ensembling was achieved at: max depth=10, n estimators=4500 (Numberof trees), learning rate=0.006, num leaves=511, min child weight=50, cols ample by tree=0.30, Using Light GBM.

VIII. Ensembling

Ensembling is the method of combining two different models and taking the best out of those models , in this assignment we used the weighted probability method since we were dealing with probabilities. We used different combinations of models but the best result that we achieved was when we used our best result on the given dataset and the best result on using oversampled data (both done using Light GBM) . The ratio of 3:1 for Given Dataset Result and Oversampled Dataset Result gave us our highest score of 0.72329.[16]

IX. Testing Method

For testing we divided the training data into 80 percent of training data and 20 percent of testing data(stratified) using the train test split from sklearn.[17] Then we used the auroc metric from sklearn library to get local scores.[18] The scores were more or less similar to the scores we got on the scoreboard.

X. Results

After Trying and tuning a lot of things, below are the best scores which we could get for each approach.

S. No.	Training Model	Public leaderboard Score	Private leaderboard Score
1.)	Random Forest Classifie	0.70276	0.70509
2.)	XgBoost Classifier	0.70969	0.71214
3.)	Light GBM classifier	0.71965	0.72241
4.)	Light GbM with random oversampleing	0.71793	0.71843
5.)	Model 3 and model 4 ensembled with weighted probability	0.72135	0.72329

XI. Conclusion

We would like to conclude that we were able to come up with an efficient model to predict whether a machine would be hit by malware or not. Such projects have a potential scope to make machines and electronic devices safe to use without any leak of data into disastorous hands.

Acknowledgment

We would like to thank our professors Professor GS Raghavan and Professor Neelam Sinha for imparting us the knowledge in the field of Machine Learning and giving us a great headstart in this world of Machine Learning. Next we would also like to thank our amazing TA's who have been teaching us all the things that we have needed for the course and also for our project and the assignments and we would especially like to thank Tejas for his constant guidance and help in clearing all the doubts that we faced throughout this past 2 weeks.

REFERENCES

[1] Microsoft *Microsoft Malware Prediction Overview*Available: https://www.kaggle.com/c/microsoft-malware-prediction/overview

[2] LongYin, Microsoft Malware Prediction Discussion https://www.kaggle.com/c/microsoft-malware-prediction/discussion/75834

[3] Zakaria Jaadi *A Step By Step Explanation Of Principal Component Analysis*[Online] Available : https://builtin.com/data-science/step-step-explanation-principal-component-analysis.

[4] ML *ML Tutorial Session 16th October*[Online] Available : https://github.com/frenzytejask98/ML_TA_IIITB_2020/tree/master/October_16

[5] Jason Brownlee Bagging and Random Forest for Imbalanced Classification [Online] Available: https://machinelearningmastery.com/bagging-and-random-forest-for-imbalanced-classification/

[6] imbalanced-learn API API documentation of the imbalanced-learn toolbox [Online] Available: https://imbalanced-learn.org/stable/api.html

[7] Tony Yiu Understanding Random Forest[Online]Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[8] XGBoost Documentation[Online]Available: https://xgboost.readthedocs.io/en/latest/

[9] Jason Brownlee A Gentle Introduction to XGBoost for Applied Machine Learning[Online]Available: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

[10] Light GBM official documentation [Online]Available: https://lightgbm.readthedocs.io/en/latest/

[11] Catboost Documentation[Online]Available: https://catboost.ai/docs/concepts/python-reference_$catboostclassifier$

[12] Sunil Ray Quick Introduction to Boosting Algorithms in Machine Learning[Online]Available: https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/

[13] Sharoon Saxena A Beginner's Guide to Random Forest Hyperparameter Tuning[Online]Available: https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

[14] Cambridge Spark Hyperparameter tuning in XGBoost[Online]Available: https://blog.cambridgespark.com/hyperparameter-tuning-in-xgboost-4ff9100a3b2f

[15] Random Search CV Documentation [Online]Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

[16] Aishwarya Singh A Comprehensive Guide to Ensemble Learning https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/

[17] Train Test split documentation https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_$test$_split

[18] Aucroc score documentation http://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_$auc_score.html$